ActionVLM - Leveraging Web-Scale Data and Instruction Tuning for Reinforcement Learning Environments

Xiangsheng Gu
Georgia Institute of Technology
xgu304@gatech.edu

Jinchu Li
Georgia Institute of Technology
jinchu.li@gatech.edu

Jack Wessell
Georgia Institute of Technology
jwessell6@gatech.edu

Chunhao Zou
Georgia Institute of Technology
czou40@gatech.edu

February 9, 2025

Abstract: Foundation models across various modalities have enjoyed unparalleled improvements in generalization over the past few years. These gains can be attributed to a variety of factors including but not limited to web-scale data, increases in parameter count, and training techniques such as instruction tuning. Despite these improvements, reinforcement learning techniques have not yet managed to achieve similar generalization across multiple environments without fine-tuning. In this project [1], we explore whether the generalization inherent to foundational vision language models can be applied to various reinforcement learning environments. Our work [2] aims to determine whether the ability of large foundational vision-language models to generalize beyond their training data can be extended to reinforcement learning by having the models act as agents, reward functions, or reward function code generators in unseen environments given a state and a goal.

Keywords: Deep Reinforcement Learning, Computer Vision, Pattern Recognition

1 Introduction

Recent advances in understanding the scaling laws that govern deep learning models have led to unparalleled performance of neural networks in a variety of modalities. Furthermore, these increases in model size, dataset size, and training times have led to an unexpected consequence - out of distribution generalization. Researchers found that LLMs trained to perform masked token prediction of web-scale datasets could out of distribution tasks with high accuracy without being trained to do so [8]. Similar examples of generalization can be seen in many of the multi-modal foundation models, and training techniques such as instruction tuning improve the ability of such models to solve novel problems.

While fields such as natural language processing, computer vision, and multi-modal modeling have all enjoyed the benefits of this generalization over the past few years, similar improvements in the field of reinforcement learning are yet to be realized. There have been attempts to create reinforcement learning models that generalize across many similar environments such as Atari games [2]. However, these models achieve mediocre performance across the environments and require fine-tuning to individual tasks/games to become competitive with smaller specialized models. This is in stark contrast to foundational vision models and language models, which achieve SOTA performance across dozens of benchmarks.

[1]**To view the video presentation:** https://drive.google.com/file/d/1DOt7qTg4eSd1xX0emaYsDl0FlCgDQPkQ/view?usp=sharing
[2]**To download the zip file of our code implementation:** https://github.com/GreysonBackup/ActionVLM_Project

We posit that this lack of generalization is a direct result of the small datasets and model architectures that are used in works such as those mentioned above. We examine whether large foundational models such as LlaVA, Claude Sonnet, GPT-4o, or other related models have sufficiently advanced reasoning, visual capabilities, and instruction following abilities to guide action generation in environments such as snake and Atari. We also investigated the feasibility of using BLIP-2 (a pre-trained VLM) as an adaptable reward mechanism, with a focus on its application in RL environments. Unlike traditional reward functions that rely solely on predefined scalar feedback, our approach combines environment-generated rewards with semantic feedback derived from BLIP-2's analysis of rendered images. Importantly, we will not be utilizing any fine-tuning throughout our work in this project - we aim to determine whether the training process these models undergo including pre-training, instruction tuning, and RLHF exposes the model to sufficient information to effectively evaluate such environments out of the box.

2 Related Work

Large-Language and Vision Assistant (LLaVA) is a more modern and powerful VLM - and one that we utilize throughout this work [4]. Unlike previous models such as CLIP which outputs embeddings [6], LLaVA is a generative model that outputs text given a text and image input. LLaVA utilizes a pre-trained LLM decoder to take advantage of its knowledge of web-scale datasets. To add image processing capabilities, LLaVA includes an image encoder that generates tokens given an image input. These tokens have the same dimensionality as the text embeddings and are simply appended to the text sequence. Unlike other VLMs, LLaVA goes through an additional training step known as visual instruction tuning. In this step, LLaVA is optimized to respond to requests from users that ask it to perform a specific task, such as describing a scene or answering questions given an image-text pair. This process makes LLaVA far-more useful to end-users as it learns how to solve specific tasks. It also allows LLaVA to solve tasks that it has not been specifically trained to do. The flexibility of LLaVA produced by its generative capabilities and instruction tuning-based training make it very attractive for solving general, out-of-distribution problems.

Instead of feeding tokens directly into a text decoder, BLIP-2 first feeds an encoded image and its aligned text tokens pair into a module known as a querying transformer [3]. The Q-former takes as input a variably sized image-text pair, but outputs a fixedThis design reduces computation and bridges the modality gap. Unlike other VLMs that use dense vision-text alignment, BLIP-2 employs a) Representation learning that enforces relevant feature extraction. b) Generative learning that adapts visual information for language tasks. This dual-stage approach aligns closely with our work, in which we adapt BLIP-2 to provide semantic rewards in reinforcement learning.

3 Methods

In our work, we utilize the generalization capabilities of large foundational models to solve reinforcement learning environments in three distinct ways:

- Foundational models to generate rewards - manual reward shaping can be an incredibly arduous process. In these experiments, we explore whether it is possible to leverage the instruction-following capabilities of foundation models to measure how close an agent is to solving an environment and whether the action it has taken will bring it closer to the goal.
- Foundational models to generate actions - policies created by existing training techniques tend to be incredibly finicky to train and lack generalization to multiple environments. Our hope is that foundational VLMs, having been both instruction tuned to solve general tasks and exposed to web-scale datasets, have sufficient problem-solving capabilities to solve reinforcement learning environments.
- Foundational models to generate executable reward function codes - where we use the LLM to directly generate executable Python codes to calculate the reward based on the environmental state, and use the experimental results to improve the reward function design iteratively.

3.1 Reward Generation

We use a deep Q-Network to estimate $Q^*(s,a)$, which represents the expected maximum return of taking action a in state s and following the optimal policy:

$$Q^*(s,a) = E_{s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

where r is the reward, γ is the discount factor, and s' is the next state.

We utilize a BLIP-2 model to analyze the rendered environment images using a prompt and outputs textual descriptions, which is translated into a reward R_v based on predefined rules (e.g., Penalizing for incorrect positioning and instability):

$$\begin{aligned}
&R_v \leftarrow R_v + 1.25, \text{ if "center" or "middle" appears.}\\
&R_v \leftarrow R_v + 2.75, \text{ if "inside" or "within" appears.}\\
&R_v \leftarrow R_v + 0.45, \text{ if "on" or "landed" appears.}\\
&R_v \leftarrow R_v + 0.75, \text{ if "stable" appears.}\\
&R_v \leftarrow R_v - 1.75, \text{ if "outside" or "away" appears.}\\
&\ldots
\end{aligned}$$

Our resulting reward function combines environment-generated rewards (R_e) with feedback rewards derived from BLIP-2 (R_v) as: $R = (1-\alpha)R_e + \alpha R_v$. we set $\alpha = 0.15$ to optimize the balance of the reward contributions. Then, the training process follows an epsilon-greedy policy for exploration. The model is trained for 300 epochs using the Adam optimizer with a learning rate of 10^{-3}. The policy is evaluated every epoch using the *LunarLander-v2* environment's native reward metric. The results are plotted to track the progress of the training and analyze the impact of the BLIP-2 feedback on the performance.

3.2 Action Generation

In our action generation experiments, we utilize a standard MDP environment structure:

$$MDP = (S, A, R, T)$$

It is paramount to remember that in these experiments no training is performed - we are exclusively examining the out-of-the-box capabilities of pre-trained foundation models to perform in reinforcement learning environments. Thus, the reward portion of the MDP, while useful for evaluation, does not effect the performance of the agent. Instead, much of the work involved is spent translating the state and action spaces between forms understandable by both the environment interface and the VLM. For our experiments, we use LLaVA one-vision as our small open-source model and Claude 3.5 Sonnet as our large proprietary model.

The state space is an output provided by the environment and can take on many forms. For example, in LunarLander-v2 the state is an 8-dimensional vector. However, in the Atari environments it is represented by an image alone. Luckily, all OpenAI gym environments contain a render function. This allows us to easily translate the state that is specific to each environment into a form that is understandable by the VLM. However, we found that all VLMs (LLaVA variants and Claude 3.5 Sonnet) often struggled with determining low-level features such as alignment or positions (this is a major hurdle we will discuss in later sections). When applicable, state information such as locations, speeds, angles etc. were embedded directly into the text prompt to aid the VLM in understanding precise features of the state space. Thus, our state space is represented as an image-text pair that contains all information necessary to solve the environment. The text prompt also includes information such as the general goal of the environment so that the agent can determine an idea of how to proceed.

The output of the VLM must be mapped onto the action space expected by the target environment. This is one major hurdle that must be overcome to apply the generalization abilities of VLMs to the specific interfaces provided RL environments. To aid in this, we first describe the relevant output space to the VLM. This helps the VLM understand what it can and cannot do in a single iteration. As always, extracting precise information from potentially long text outputs is difficult. To overcome this, we found success with the two following techniques:

- Few-shot grounding - we provide an example (image, text) and response pair in which we write the response. The response is in the format we expect the model to respond with, and the model uses the provided examples as a template.
- Specific format request - for larger and smarter models like Claude 3.5, we found it sufficient to request that the model output its decision in a specific format, such as enclosed in parantheses or single quotes. While smaller models like LLaVA struggled with these instructions, it worked seamlessly with stronger models and was our preferred method

By converting the state and action spaces like so, we give our models the capability to interact with general environments with only a few limitations that will be discussed in later sections. For these experiments, we focused on the snake-v0 and LunarLander-v2 environments. We had originally planned to work with Atari environments like breakout as well, but due to limitations we will discuss later we had to abandon these experiments.

3.3 Reward Function Code Generation

This approach closely follows from the Eureka [5] paper, where we use LLMs to generate and iteratively improve the reward function codes. The authors introduced two innovative in-context code improvement steps: evolutionary search, where multiple candidates are generated within the same iteration and the code with the highest success rate is used as input to the next iteration, and reward reflection, where the LLM needs to reflect on the experimental results of the previously generated code to propose new modifications to the code. We adapted the original Eureka code and removed its dependencies on IsaacGym and TensorBoard. Instead, we used the OpenAI Gymnasium environment and Stable-Baselines3 for our experiments. Specifically, we utilized a modified CartPole environment, where the objective is to make the cart balance the pole while moving back and forth periodically as quickly as possible. We conduct 7 iterations in total. During each iteration, 3 candidate reward function codes are generated, and we run experiments on all 3 candidates using the PPO algorithm with default hyperparameters and a training time step of 300000. Every 10000 steps, we record the reward component breakdowns, the overall GPT-generated rewards, and the fitness function by running evaluations on the model with 5 rollouts. The fitness function is a sparse ”ground-truth reward” that consists of two components: a survival score, which is 1 for every time step, and a movement score. As long as the cart enters a ”threshold” (left boundary or right boundary) position, a reward of 10 given. If the cart completes one swing back and forth, a reward of 100 is given. Given that the maximum time step of the environment is 2000, a score above 2000 means the agent can balance the pole while moving back and forth. After all 3 candidates finish training, the best candidate code is selected based on the fitness function and proceeds to the next round. In the next round, this code is fed into the prompt so GPT-4o can propose modifications for iterative improvement.

In this paper, we explore the importance of visual grounding by testing whether feeding screenshots from sample rollouts into the prompts improves the generation of reward function codes compared to without visual input. For this purpose, we capture 5 screenshots from a sample rollout at specific time frames: the first frame, the 25% time frame, the midpoint frame, the 75% time frame, and the final frame. We ask the LLM to analyze the agent's behavior based on the visual input in addition to the reward breakdown data proposed by the original Eureka paper. Harnessing the novel modalities of the latest LLMs is one of the key contributions and innovations of our paper.

4 Experimental Results

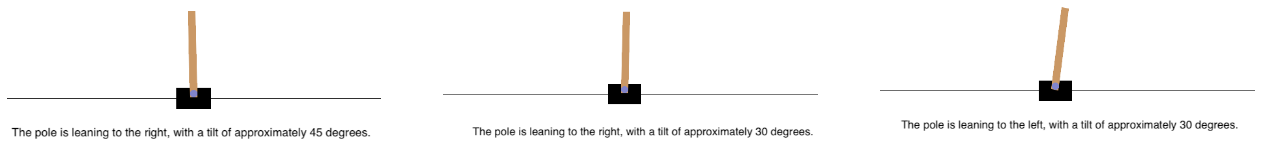
4.1 Reward Generation — LLaVA + Deep Q-Network

In our initial experiments, we integrated the LLaVA (Large Language and Vision Assistant) model as a reward function within the *CartPole-v1* environment. The primary objective was to assess the model's capability to accurately interpret the pole's tilt degree from rendered frames and translate this visual information into meaningful reward signals for the Deep Q-Network (DQN) agent.

We employed the following prompt:

```
"The image is about a pole sitting on a cart. Is the pole leaning left or right? Output
```

```
the degree that it’s leaning. Round the degree to the nearest whole number."
```


The pole is leaning to the right, with a tilt of approximately 45 degrees.

The pole is leaning to the right, with a tilt of approximately 30 degrees.

The pole is leaning to the left, with a tilt of approximately 30 degrees.

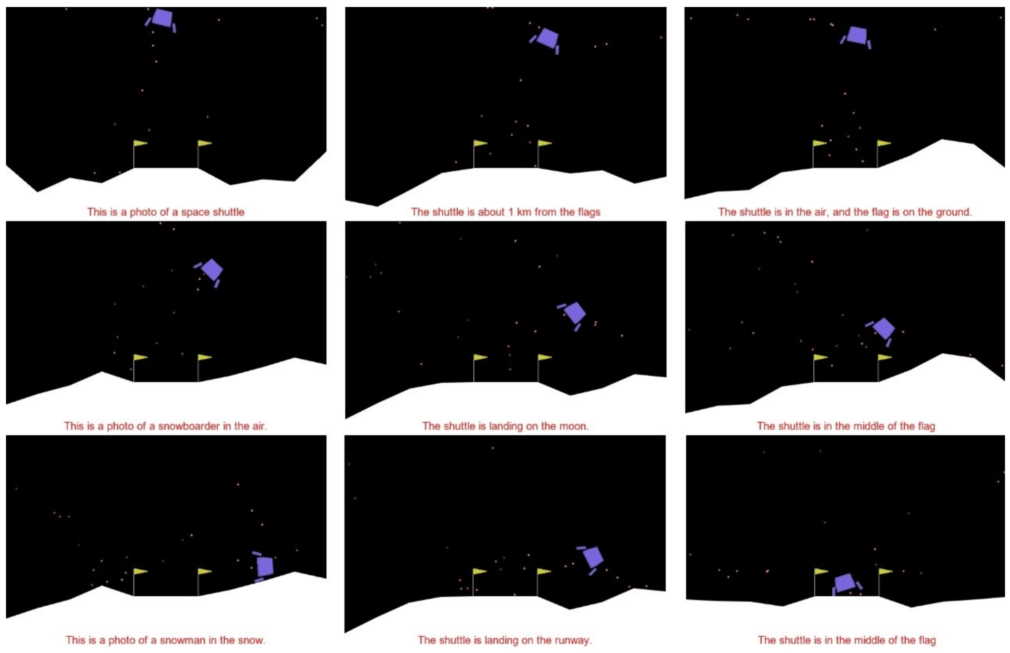
Figure 1: Sample frames and outputs using LLava-7b

Throughout the experiments, we observed that the LLaVA model consistently reported the pole’s tilt degree as exactly 45 or 30 degrees, irrespective of the actual angle depicted in the environment’s rendered images. This uniform output occurred even when the pole was visibly leaning significantly more or less than the reported degree to either side. Such behavior indicates a lack of nuanced understanding and adaptability in the model’s visual interpretation capabilities within this specific task.

From these observations, we conclude that smaller-scale models like LLaVA 7B are currently inadequate for precise reward generation tasks that require detailed visual interpretation and quantitative reasoning. The inability to dynamically assess and report varying degrees of pole tilt suggests that more sophisticated or larger models may be necessary to achieve reliable and meaningful reward signals in reinforcement learning environments.

4.2 Reward Generation — BLIP-2 + Deep Q-Network

We conducted several experiments in terms of adjusting prompts to capture better frame descriptions of the *LunarLander-v2* environment. Over dozens of prompts are tested and we finally decided to use three distinct prompts: 1) Tell me what is happening in this picture. 2) In this photo, how far in km is the purple space shuttle in relation to the yellow flags? 3) Analyze the landing situation of the space shuttle in terms of its position relative to the snow surface inside the flags.


This is a photo of a space shuttle

The shuttle is about 1 km from the flags

The shuttle is in the air, and the flag is on the ground.

This is a photo of a snowboarder in the air.

The shuttle is landing on the moon.

The shuttle is in the middle of the flag

This is a photo of a snowman in the snow.

The shuttle is landing on the runway.

The shuttle is in the middle of the flag

Figure 2: Images with generated texts at the bottom across the three columns corresponding to three different prompts

For the first prompt, the model primarily provided generic descriptions, indicating an interpretive tendency to misclassify or oversimplify objects when queried generically. For the second prompt, the generated responses sometimes can be quantitative, which revealed inconsistencies in metric assessment and positional reasoning. For the third prompt, output answers were relative and spatial, aligning better with contextual positioning but still include variability in interpretation.

The baseline DQN required approximately 45 minutes to train for 300 epochs and achieve cumulative rewards exceeding 200, which is the threshold for solving the *LunarLander-v2* environment. Meanwhile, the BLIP-2 integrated DQN took approximately 200 minutes more to train the same amount of epochs, which is due to the additional computation involved in the feedback generation.

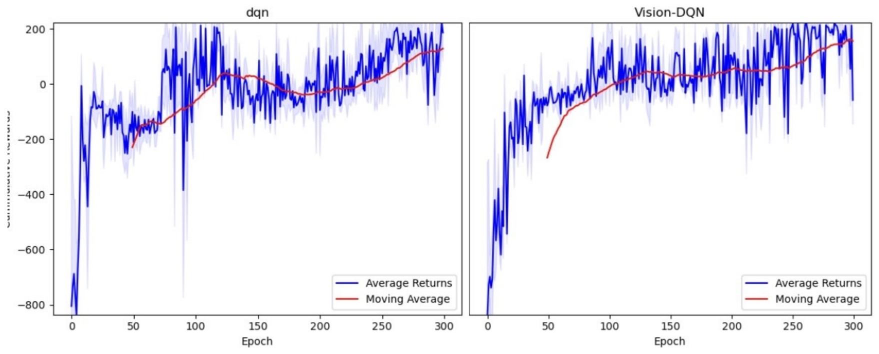


Figure 3: Plots of the cumulative reward curves for both the baseline DQN (left) and the BLIP-2 integrated DQN (right)

In Figure 3, the reward curve for the BLIP-2 integrated DQN exhibited a relatively smoother increase over epochs. This may suggest that the semantic feedback improved the consistency and stability of the learning process.

4.3 Action Generation — LLaVA and Claude 3.5

In this section, we will discuss our results from using LLaVA and Claude as policies to solve basic RL environments.

4.3.1 LLaVA

In our experiments, LLaVA-7b was unable to understand the instructions we had given it. In a zero-shot setting it would output nonsensical responses, performing on-par with taking random actions in each environment. In a few-shot environment, it would often default to mimicry, outputting the same information given to it in the example prompts with little-to-no reasoning being apparent. After brief experiments with LunarLander and snake, we determined trying to achieve meaningful levels of understanding with LLaVA would be futile. In conclusion, we suspect that LLaVA-7b is simply too small to understand the relationships between the image, the goal we provide, and how its actions may affect the environment. We had hoped to work with LLaVA-72b as a middle-ground between LLaVA-7b and Claude, but were unable to receive a GPU cluster large enough to fit the model (even in 8-bit precision it was too large to fit on a single A100).

4.3.2 Claude 3.5 Sonnet

The results from the Claude model are far-more interesting. The paid nature of utilizing the Claude API limited our ability to run robust experiments. However, we found that Claude possesses a latent ability to synthesize an instruction, a state, and a goal that causes it to exhibit surprising zero-shot performance on unseen tasks.

On the snake environment, Claude demonstrates a surprising ability to understand the goal of the game and how its potential outputs will affect the board. It effectively avoids the walls of the board and understands that it must eat food

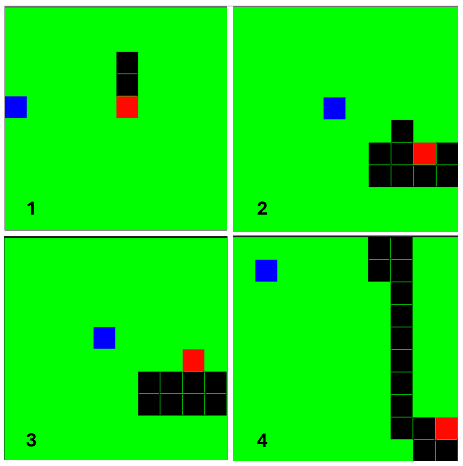
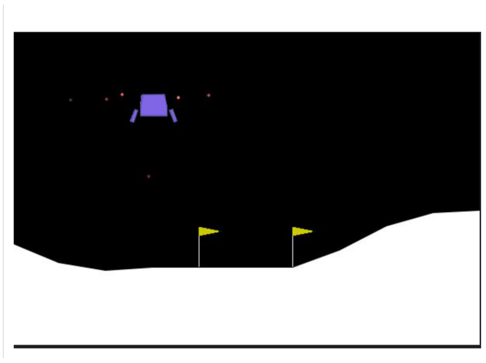


Figure 4: Selected frames from a game of snake played by Claude. 1: The starting state. 2 and 3: nearby frames from the middle of the game - Claude is able to safely avoid its tail. 4: The second to last frame - at this move, Claude does not see its tail and turns into it, losing the game.

to gain points. Again, due to the paywall that Claude is locked behind, we were limited in the number of experiments we could run. At its longest, Claude was able to grow the snake to an impressive length of 14 units by eating 10 foods while avoiding its tail and staying within the boundaries of play. It seems to be limited by its inability to understand fine-grained object relationships and plan in advance - Claude would often get itself into precarious positions then run into its own tail. Figure 4 shows a variety of frames from an example game including a transition state and a failure case. Although Claude is unable to solve the environment, its ability to synthesize instructions, a state, and a goal despite never being trained to do so is impressive nonetheless.



```
Let's analyze the situation:
1. Position: You're slightly left of center
which is good as you're close to the landing
zone 2. Horizontal speed: Moving left at 0.55,
which needs to be slowed down 3. Vertical speed:
Falling at 0.25, which is manageable but should
be monitored 4. Rotation: Small counterclockwise
rotation with a nearly horizontal angle. Given
these conditions: - The leftward drift needs to
be countered - The descent speed is acceptable
but shouldn't increase much more - The
counterclockwise rotation should be stopped
Since your ship is almost horizontal and moving
left, firing the main thruster would help slow
your leftward movement while also providing some
lift to manage the descent speed. The slight
counterclockwise rotation isn't severe enough to
prioritize correcting it over managing your
horizontal velocity. 'fire main'
```

Figure 5: An example input-output pair from about half-way through a run. Clearly, Claude can keep the ship stable but struggles to understand the impact some of its decisions may have.

On the LunarLander-v2 environment, Claude again exhibits an impressive ability to understand what actions it must take to achieve its goal. We found that Claude often struggled with determining what to do given only an image, the goal, and a description of the action space. However, when provided with precise descriptions of how the lander is moving, Claude was able to keep the spacecraft level while descending gently toward the surface. Figure 5 includes an example input image and response from Claude - note how its reasoning is correct but cannot perfectly determine how its actions will impact the state. Despite the impressive behavior, as the lander nears the surface Claude was never able to effectively land it on the surface. Similar to the Snake environment, it seems that Claude's ability to distinguish between objects diminishes as they become closer and closer together. So, although it never crashes the lander, it does not ever successfully land it either and instead the lander eventually drifts out of frame.

In this section, we did not include any comparisons with baselines and included only limited numerical results. This is for the following two reasons:

- There is no fair open-source comparison with which to compare Claude to. It has never been trained as a controller, and most existing models or methods are specialized to specific environments.
- Claude's performance on the environments is still objectively poor - even basic methods like vanilla Q-iteration and actor-critic can solve environments like Snake and LunarLander-v2 as we saw in the first homework.

Because of our inability to run robust experiments without spending large sums of money (sending images over the Claude API is far-more expensive than just text) we likely have not fully-realized the full potential of Claude and similar models. Despite its objectively poor performance, Claude's ability to grasp the state, goal, and result of its decisions on a task that is so different from its training data is impressive nonetheless. We will go into more detail on its limitations and how they impact its performance in the following section.

Note: We originally planned to evaluate Claude on Atari environments as well. However, its poor performance on simpler environments coupled with its reliance on information being embedded into the prompt suggested to us that evaluating it on such complex environments would be futile.

4.4 Reward Function Code Generation — GPT 4o

We conducted 4 experiments using 2 environments, the original CartPole environment, where the goal is to simply balance the pole, and the modified CartPole environment, where the goal is not only to balance the pole but also to move the cart back and forth as quickly as possible. For the modified CartPole environment, we introduced 3 additional variables in the game state:

- swing_threshold: A constant defining the range the cart must move back and forth between.
- has_touched_left_threshold: A flag indicating whether the cart has reached the left swing threshold.
- has_touched_right_threshold: A flag indicating whether the cart has reached the right swing threshold.

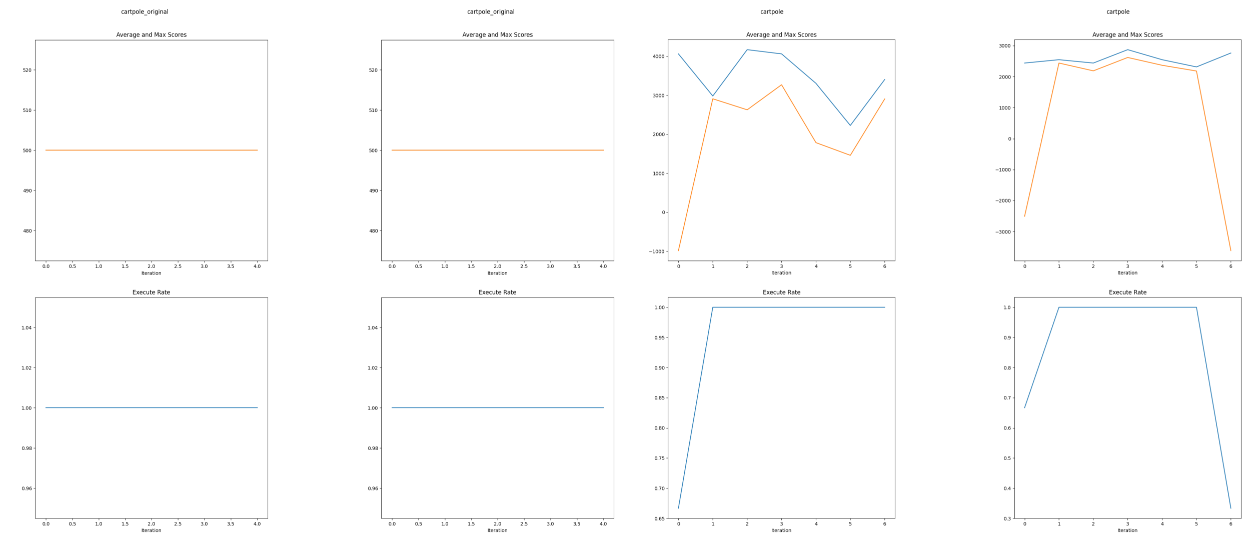
For the simpler task, both approaches (including both reward changes and screenshots in the prompt as feedback versus not including screenshots in the prompt) consistently resulted in desirable agent behavior (balancing the pole for 500 time steps), However, for the more complex task of balancing and swinging, our experiments (Figure 6) show that GPT-4o is more likely to generate reward function codes that produce desirable results when screenshots are included in the prompt. A demonstration is in the video presentation. When screenshots of sample rollouts are included in the prompt, the best candidate in each iteration achieves a ground-truth score higher than 2000, indicating successful pole balancing as well as additional rewards for surpassing the swing threshold. Notably, 5 out of 7 iterations achieve a score above 3000, and 3 out of 7 iterations exceed a score of 4000, indicating that more rewards from the cart's movement are collected. In contrast, none of the 7 candidates achieve a score above 3000 when image input is not provided.

5 Discussion and Analysis

5.1 Reward Generation — BLIP-2 + Deep Q-Network

Our approach shown that certain prompts can yield more contextually relevant responses, thus task-specific fine-tuning or prompt engineering are needed in order to enhance interpretative accuracy for VLMs in RL tasks. Our original goal was to determine if large multi-modal models could act as the sole reward function exclusively. However, this proved fruitless given LLaVA and Blip-2's capabilities and had to fall back on using a linear combination of generated and environment rewards.

- **Strengths:** DQN's learning stability is slightly improved with the integration of BLIP-2.
- **Weaknesses:** Generalization errors often happen, which indicates future improvements are needed in visual understanding and semantic alignment. BLIP-2 integration significantly raised computational costs, which limited the applicability to larger-scale environments.



(a) Original CartPole, With Visual Input

(b) Original CartPole, Without Visual Input

(c) Modified CartPole, With Visual Input

(d) Modified CartPole, Without Visual Input

Figure 6: Original and Modified CartPole Experiments w/ and w/o Visual Input

5.2 Action Generation - LLaVA and Claude 3.5

As mentioned in the results section, smaller models like LLaVA are incapable of understanding even the simplest of reinforcement learning tasks such as snake. Thus, we will limit our discussion to our results with Claude 3.5 which can likely be applied to other large proprietary models.

1. VLMs are blind: As discussed earlier and in [7], although models like Claude struggle at understanding fine-grained features such as intersecting lines, object orientations, or detecting small objects. This skill is paramount for performance in many RL environments.
2. Spatio-temporal reasoning: Although proprietary models like Claude and ChatGPT have very limited understanding of video data [1]. This lack of an ability to understand relationships between video frames is a major bottleneck - these models are unable to fully understand how their actions will effect the state of the environment.
3. Prompt Engineering and Discrete Action Spaces: Prompt engineering was the primary time sink in this portion of the project. This obviously calls into question whether the model is truly generalizing. Additionally, the current method is limited to discrete action spsces which limits generalization.

6 Conclusion

Although we explored the topic of utilizing foundational VLMs from two different perspectives, the results of both our reward and action generation experiments point to the same conclusions. From both sets of experiments, we found that although the ability of these models to generalize to RL environments improves with the scale of the model, they still lack the reasoning abilities required by such tasks. Our work shows that utilizing such models as general agents in RL environments is a promising direction to take future research. The primary roadblock that current models face is their inability to distinguish fine-grained visual features and understand how the actions that they may take will impact the environment. If future large multi-modal models can solve these issues, it is highly likely that they may be useful as agents in general environments.

References

[1] Anthropic. Claude 3.5 sonnet model card addendum, 2024. URL https://www-cdn.anthropic.com/fed9cc193a14b84131812372d8d5857f8f304c52/Model_Card_Claude_3_Addendum.pdf.

[2] K.-H. Lee, O. Nachum, M. Yang, L. Lee, D. Freeman, W. Xu, S. Guadarrama, I. Fischer, E. Jang, H. Michalewski, and I. Mordatch. Multi-game decision transformers, 2022. URL https://arxiv.org/abs/2205.15241.

[3] J. Li, D. Li, S. Savarese, and S. Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 2023. URL https://arxiv.org/abs/2301.12597.

[4] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning, 2023. URL https://arxiv.org/abs/2304.08485.

[5] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models, 2024. URL https://arxiv.org/abs/2310.12931.

[6] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

[7] P. Rahmanzadehgervi, L. Bolton, M. R. Taesiri, and A. T. Nguyen. Vision language models are blind, 2024. URL https://arxiv.org/abs/2407.06581.

[8] J. Treutlein, D. Choi, J. Betley, S. Marks, C. Anil, R. Grosse, and O. Evans. Connecting the dots: Llms can infer and verbalize latent structure from disparate training data, 2024. URL https://arxiv.org/abs/2406.14546.